DAG-width and circumference of digraphs

Jørgen Bang-Jensen and Tilde My Larsen[*]

February 12, 2015

Abstract

We prove that every digraph of circumference l has DAG-width at most l and this is best possible. As a consequence of our result we deduce that the k-linkage problem is polynomially solvable for every fixed k in the class of digraphs with bounded circumference. This answers a question posed in [2]. We also prove that the weak k-linkage problem (where we ask for arc-disjoint paths) is polynomially solvable for every fixed k in the class of digraphs with circumference 2 as well as for digraphs with a bounded number of disjoint cycles each of length at least 3. The case of bounded circumference digraphs is open. Finally we prove that the minimum spanning strong subdigraph problem is NP-hard on digraphs of DAG-width at most 5.

Keywords: DAG-width, k-linkage problem, bounded cycle length, polynomial algorithm, cops and robber game

1 Introduction

Terminology and notation not described below follows [1]. A digraph $D = (V, A)$ has vertex set V and arc set A. The **out-degree** (**in-degree**), denoted $d^+(v)$ ($d^-(v)$), of a vertex v is the number of arcs from v to $V - v$ (from $V - v$ to v). For $X, Y \subset V$ with $X \cap Y = \emptyset$ an (X, Y)-path is a directed path starting in X and ending in Y and with all other vertices in $V - (X \cup Y)$. For a directed path or cycle P containing vertices u, v with u before v on P we denote by $P[u, v]$ the subpath of P from u to v. We use the notation $[k]$ for the set $\{1, 2, \ldots, k\}$. A k-cycle is a directed cycle with k vertices. The **circumference** of a digraph D is the length of a longest directed cycle in D. A directed acyclic graph (DAG) is a digraph without directed cycles.

For a given natural number k the k-linkage problem is as follows: Given a digraph D and $2k$ distinct vertices $s_1, \ldots, s_k, t_1, \ldots t_k$ (called **terminals**); determine whether D has k disjoint paths $P_1, \ldots, P_k$ such that P_i is an (s_i, t_i)-path for $i \in [k]$. While the undirected analogue of the k-linkage problem is polynomial for every fixed k by the Robertson-Seymour theorem [15], the directed version is NP-complete already for $k = 2$ [7]. The problem is known to be polynomially solvable for fixed k when D belongs to one of the following classes of digraphs: acyclic digraphs [7], semicomplete digraphs [3, 6], digraphs of bounded directed-tree-width [12] (this includes digraphs of bounded DAG-width which will be defined later), digraphs of bounded Kelly-width [11] and d-path-dominant digraphs [6]. A digraph D is d-path-dominant for some $d \geq 1$ if every minimal path[1] P of D with d vertices has the property that there is at

---

[*]Department of Mathematics and Computer Science, University of Southern Denmark, Denmark

[1]A path P from a vertex x to a vertex y is **minimal** if there is no (x, y)-path P' such that $V(P')$ is a proper subset of $V(P)$.

least one arc between every vertex of $D - V(P)$ and $V(P)$. Thus the 1-path-dominant digraphs are the semicomplete digraphs (every pair of distinct vertices have at least one arc between them).

In [2] it was asked whether the k-linkage problem would be polynomially solvable for digraphs of bounded circumference (see also [10]). This can be seen as a generalization of the result by Fortune et al for acyclic digraphs. In this paper we answer the question in the affirmative by showing that digraphs of bounded circumference have bounded DAG-width (defined below) and hence also bounded directed tree-width. From this the result follows since the k-linkage problem is polynomial for digraphs of bounded directed tree-width [12].

We also consider the weak k-linkage problem where we ask for arc-disjoint paths $P_1, \ldots, P_k$ such that P_i is an (s_i, t_i)-path for $i \in [k]$. Now we may have $|\{s_1, \ldots, s_k, t_1, \ldots, t_k\}| < 2k$ (e.g. $s_2 = s_6 = t_1$). We prove that for every fixed natural number k the weak k-linkage problem is polynomially solvable for digraphs of circumference 2, digraphs with no closed trail longer than some constant and for digraphs having no set of l disjoint cycles each of length at least 3 (the proof of the latter uses a result from [10], see Theorem 5.2).

2 DAG-width: definitions and some results

Robertson and Seymour introduced the concept tree-width of undirected graphs. The tree-width measure has many nice properties, including polynomial solutions for many NP-complete problems on graphs of bounded tree-width. Several attempts have been made to find a measure for directed graphs with similar properties as tree-width for undirected graphs. Unfortunately there is evidence [9] that in some sense none can exist, but several measures that work nicely on certain problems have been made. In the following we will consider one of these, the DAG-width of a directed graph. The theory of this section is based on [14, 4]. We will start by defining the concept of DAG-width and then relate this to a directed version of the cops and robbers game. As the name suggests, the DAG-width of a digraph is a measure of how close it is to being acyclic.

Definition 2.1. For an acyclic digraph $D = (V, A)$ we define the partial ordering $\leq_D$ to be the reflexive, transitive closure of the arcs of D. A **root** of a set $X \subseteq V$ is then a $\leq_D$-minimal element of X and analogously a **leaf** of a set $X \subseteq V$ is a $\leq_D$-maximal element.

Definition 2.2. Let $D = (V, A)$ be a directed graph with vertex set V and arc set A. We say a set $W \subseteq V$ **guards** a set $V' \subseteq V$ if every arc leaving V' is incident with W, i.e. for all $uv \in A$ with $u \in V'$ and $v \notin V'$ we will have $v \in W$.

We can now define the DAG-width.

Definition 2.3. [14] Let $D = (V, A)$ be a directed graph. Then a **DAG-decomposition** of D is a pair $(H, \mathcal{X})$, where H is a DAG and $\mathcal{X} = (X_h)_{h \in V(H)}$ such that

D1) $\bigcup_{h \in V(H)} X_h = V(D)$

D2) For all vertices $h, h', h'' \in V(H)$ such that $h \leq_H h' \leq_H h''$ we have $X_h \cap X_{h''} \subseteq X_{h'}$

D3) For all arcs $(h, h') \in A(H)$, $X_h \cap X_{h'}$ guards $X_{\geq h'} \backslash X_h$ where $X_{\geq h'} = \cup_{h' \leq_H h''} X_{h''}$. For any root h, the set $X_{\geq h}$ is guarded by $\emptyset$.

The **width** of the DAG-decomposition $(H, \mathcal{X})$ is defined as $\max\{|X_h| : h \in V(H)\}$ and the **DAG-width** of D is the minimum width over all DAG-decompositions of D. The DAG-width of an acyclic graph is 1 (just let $V(H) = V(D)$ and $X_h = \{h\}$ for each $h \in V(H)$).

Notice that given a DAG-decomposition $(H, \mathcal{X})$ of D which does not have a unique root, we can always add the empty set to $\mathcal{X}$ and a corresponding new vertex to H, such that the new vertex has an arc to every root of H. This will give a new DAG-decomposition of D, with the same width as $(H, \mathcal{X})$, but with a unique root. Hence we may always assume that given a DAG-decomposition of D, it has a unique root. It can also be shown that the DAG-width of a digraph is equal to the maximum of the DAG-widths of its strongly connected components [14].

Theorem 2.4 ([14])**.** *It is NP-hard to decide for inputs D and k whether the DAG-width of D is at most k.*

We will now give two results that are related to another measure, called the directed tree-width [12]. As the definition is quite technical and not needed for our purposes, we shall not give it here.

Theorem 2.5 ([4])**.** *If a graph has DAG-width k then it has directed tree-width at most $3k + 1$.*

Theorem 2.6 ([12])**.** *For every fixed natural number k the k-linkage problem is polynomially solvable for directed graphs of bounded directed tree width.*

Corollary 2.7 ([14])**.** *For every fixed natural number k the k-linkage problem is polynomially solvable for directed graphs of bounded DAG-width.* □

A useful way to obtain a DAG-decomposition of a given digraph is via the game of cops and robber. This game was first introduced by Seymour and Thomas and used in the study of the tree-width in undirected graphs. For the directed case, different variations of the game have been studied. We will pose the game in the context of the DAG-width [14]. The principle of the game is that a robber is moving around the digraph from vertex to vertex respecting the orientation of the arcs. The robber can run infinitely fast and wants to avoid getting caught by the cops. He can take any path from his current vertex to another, provided no intermediate vertex on that path is currently occupied by a cop. The cops can either stand on vertices of the digraph or be in a helicopters above the graph (the point of the helicopters is that cops are not constrained to move along paths in the digraph). The cops win if they can land on the vertex occupied by the robber. We say that the cops have a **cop-monotone** strategy if the cops never visit a vertex in the graph more than once. Similarly, the cops have a **robber-monotone** strategy if the the set of vertices the robber can reach (without running through a cop hosting vertex) is non increasing.

Lemma 2.8. *[4]) If the cop player has a cop-monotone or robber-monotone winning strategy then he also has a winning strategy that is both cop- and robber-monotone.*

The strategy for the cops will be to split the graph into strongly connected components in such a way that the set of vertices reachable by the robber (without running into a cop) becomes smaller and smaller until the cops can finally land on the vertex occupied by the robber. Before stating the essential connection between DAG-width and the game of cops and robber, it is helpful to consider the connection between guarding sets and the game of cops and robber. If a robber is in a set V' and there is a cop on every vertex in W for some guarding set W of V', then the robber cannot leave V' without running to a
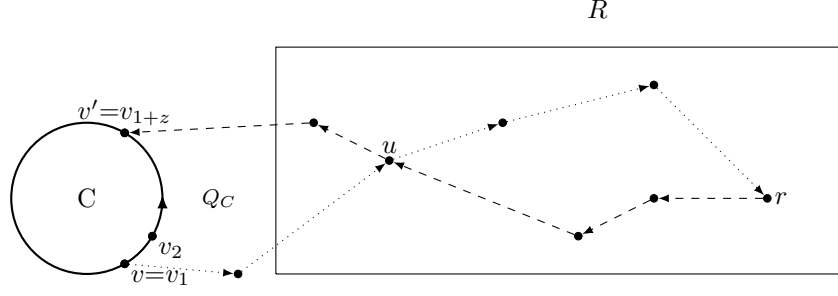
Figure 1: dotted arcs form the $P_{C,r}$ path and dashed the $P_{r,C}$ path. The cycle Q_C is formed by the three subpaths $P_{r,C}[u, v_{z+1}], C[V_{z+1}, v_1], P_{C,r}[v_1, u]$.

vertex with a cop and hence getting caught. Now if there are cops not placed on any vertex, these can land on vertices in V' and hence forcing the robber into a smaller piece of the digraph.

Theorem 2.9. *[4, 14] A directed graph D has DAG-width k if and only if it takes k cops to catch the robber using a cop-monotone strategy.*

3 Digraphs with bounded circumference

Using the game of cops and robber we are now ready to prove our main result. A cycle C is **maximal** in D if D has no cycle properly containing the vertices of C. Note that in a digraph with bounded circumference one can check maximality of any cycle C and find a larger cycle containing $V(C)$ if one exists in polynomial time.

Theorem 3.1. *Let D be a directed graph with circumference at most $p \in \mathbb{N}$. Then the DAG-width of D is at most p and this is best possible.*

Proof. Since the DAG-width of a digraph is the maximum of the DAG-widths of its strong components, it suffices to consider the case when D is strong.

Below we describe a robber-monotone winning strategy for p cops. We start by having no cops in the digraph. Now pick an arbitrary maximal cycle C of D and put $|C|$ cops on the vertices of C. If the robber is not caught already, he will have moved to a vertex r of $V - V(C)$. Let us denote by $R \subseteq V - V(C)$ the set of vertices the robber can reach without running into a cop, that is the set of vertices reachable from r in $D - V(C)$. As D is strong, there must be a $(V(C), r)$-path $P_{C,r}$ starting in some vertex v of C and ending in r. Similarly, there must be an $(r, V(C))$-path $P_{r,C}$ from r to some vertex v' of C. Their concatenation $P_{C,r}P_{r,C}$ will be a (possibly closed) (v, v')-trail in D. Now let u be the first vertex on the trail with $u \in V(P_{C,r}) \cap V(P_{r,C}) - V(C)$ (such a vertex exists as r is on both paths). Then we let $Q_C = P_{C,r}[v, u]P_{r,C}[u, v']$ be the path (a cycle when $v = v'$) in D formed by following $P_{C,r}$ to u and then proceeding from u to v' on $P_{r,C}$. Note that $u \in R$ as u can be reached by r in $V - V(C)$. See Figure 1.

Let us first consider the case where there exists a pair of paths $P_{C,r}$, $P_{r,C}$ such that Q_C is a path (that is, $v \neq v'$). Pick the paths $P_{C,r}, P_{r,C}$ such that $|V(C[v, v'])| > 0$ is as small as possible and let $v_1, v_2, \ldots, v_{|C|}$ be an enumeration of the vertices in C such that and $v_1 = v$ and $v_i v_{(i+1) \mod |C|}$ is an arc in C for $i \in [|V(C)|]$. Let z be the integer such that $v' = v_{1+z}$ and note that, by our choice of paths, v_{1+z} is the first vertex on the cycle C after v_1 that is the end vertex of some $P_{r,C}$ path. Now $C' = C[v_{1+z}, v_1]Q_C$ is a cycle, containing at least one vertex from R (namely u). Observe that $z \neq 1$,

4

since otherwise the cycle C' would be longer than C, contradicting the maximality of C. Move (some of) the cops currently on the vertices of $C[v_2, v_z]$ to the currently unoccupied vertices of C'. Further if there are still unoccupied vertices on C' (when $|V(C[v_2, v_z])| < |V(Q_C)|$), then place cops currently in helicopters on these. Conversely, if there are more available cops from $C[v_2, v_z]$ than needed to cover the C' vertices (when $|V(C[v_2, v_z])| > |V(Q_C)|$), then place these in helicopters for later use. Notice that we will never run out of cops, since all our placed cops are on the cycle C', and $|C'| \leq p$. Now extend C' to a maximal cycle C^* containing all the vertices of C'. After covering possible new vertices in C^*, by cops from helicopters, the strategy can be repeated starting with cops on $V(C^*)$ and we will have $R^* \subsetneq R$, where R^* is the set of vertices reachable by the robber after removing $V(C^*)$.

It remains to consider the case where the paths $P_{C,r}$ and $P_{r,C}$ are incident to the same vertex v in C for all choices of such paths, and hence every Q_C a cycle. In this case, every cop except the one in v is free to be lifted as every path from R to C enters C in v. Take an arbitrary Q_C cycle, and as in the case above, find a maximal cycle C^{**} containing the vertices of Q_C and occupy each of these vertices with a cop. Again since the length C^{**} is at most p, we have enough cops. Now we can repeat the strategy above starting from the new maximal occupied cycle.

The strategy described above is indeed a robber-monotone strategy: When moving the cops from C to C', we only move cops that cannot be reached by the robber without the robber running through another cop occupied vertex, hence we do not open up a new part of the graph for the robber, and hence $R' \subseteq R$ will always hold. Furthermore, for each new maximal cycle we occupy in the strategy, at least one of these vertices belongs to the current set R. Hence in at most $|V|$ steps $R = \emptyset$ and the robber is caught so the strategy above is a winning one.

To see that p cops may be necessary to catch a robber in a digraph with circumference p consider the complete digraph on p vertices, that is, there is an arc on both directions between any pair of distinct vertices. If we only have $p - 1$ cops here there will always be a free vertex and the robber (who moves infinitely fast) can move directly to that vertex from his current position as soon as the cops have announced their new positions. $\qquad\square$

Corollary 3.2. *For every natural number p there exists an algorithm $\mathcal{A}_p$ which given a digraph of circumference p outputs a DAG-decomposition of width p in polynomial time.*

Proof. This follows from the fact that we can translate a robber monotone strategy for p cops into a DAG-decomposition of width at most k [14]. See an example in the appendix at the end of the paper. Note that the steps in the proofs above, such as finding a maximal cycle containing a given set X of vertices, are all polynomial since the length of the cycle sought is at most p and hence we may check all possible cycles covering X in polynomial time. $\qquad\square$

Combining Theorem 2.5 and Theorem 3.1 we obtain the following.

Corollary 3.3. *Every digraph of circumference most l has directed tree-width at most $3l + 1$.*

Corollary 3.4. *[5]. If an undirected graph G has circumference at most k, then its tree-width is at most $k - 1$.*

Proof. Given G we form the digraph $\overset{\leftrightarrow}{G}$ by replacing each edge of G by a 2-cycle. Then the tree-width of G equals the DAG-width of $\overset{\leftrightarrow}{G}$ minus one [4] and now the claim follows from Theorem 3.1. $\qquad\square$

4 Linkings in digraphs with bounded circumference

The following direct consequence of Corollary 2.7 and Theorem 3.1 answers a question in [2] in the affirmative.

Theorem 4.1. *For every choice of natural numbers k, l there exists a polynomial algorithm for the k-linkage problem on digraphs with circumference at most l.* $\qquad\blacksquare$

The case $l = 2$ was proved previously in [10]. That proof uses both the polynomial algorithm for k-linkage in undirected graphs from [15] and an algorithm similar to that used in the algorithm for acyclic digraphs [7] to obtain an algorithm of running time roughly $O(n^{2k})$ where n is the number of vertices in the digraph.

The reduction below to the k-linkage problem for acyclic digraphs leads to a faster algorithm since the complexity of the overall algorithm will be the same as the complexity of solving k-linking in DAGs which is $O(k!n^{k+2})$ (see e.g. [1, Section 10.4]).

Theorem 4.2. *Let D be a directed graph of circumference 2 and let k be a fixed integer. Then the k-linkage problem in D can be reduced to a k-linkage problem in an acyclic graph in linear time.*

Proof. First observe that since we are looking for disjoint paths, we may assume that $d^-(s_i) = d^+(t_j) = 0$ for $i, j \in [k]$. Also note that each strongly connected component of D will correspond to a tree T in the underlying undirected graph where each edge uv of T is replaced by a 2-cycle on u, v.

We will transform an instance $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ to an equivalent instance $[D', s_1, \ldots, s_k, t_1, \ldots, t_k]$ where D' is an acyclic digraph by removing one 2-cycle at a time, until there are no cycles left in the graph. The terminals stay the same throughout the reduction. Let S be an arbitrary non-trivial strong component of D (if none exists, D is acyclic and we are done). By the assumption above, S does not contain any of the vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$ and hence none of the paths in a k-linkage can start or end in S.
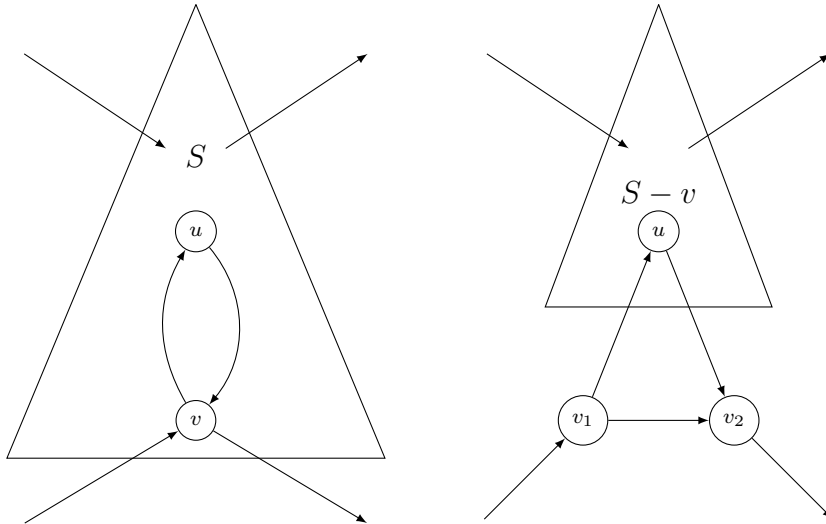

Figure 2: S is the strong component in D containing the 2-cycle uv

By the description of the strong components above, S contains a 2-cycle uvu such that v has in- and out-degree one in S. Let D^* be obtained from D by deleting v, adding two new vertices v_1, v_2, adding an arc wv_1 for each arc wv with $w \neq u$ in D, adding and arc v_2h for each arc vh with $h \neq u$ in D and finally adding the arcs v_1u, uv_2, v_1v_2, see Figure 2. It is easy to check that D^* has circumference at most 2 and that D^* has the desired paths if and only if D does: the vertex v can be part of at most one path in a solution in D and at most one path in a solution for D^* can intersect the set $\{v_1, v_2\}$.

Now it is clear that by repeatedly processing one leaf vertex at a time from a non-trivial strong component, as long as one exists, we obtain an equivalent acyclic instance after at most $|V(D)| - 2k$ steps. $\square$

As mentioned in the introduction, the weak k-linkage problem is the arc-disjoint version of the k-linkage problem, where we ask for arc-disjoint rather than vertex-disjoint paths $P_1, \ldots, P_k$ such that P_i is an (s_i, t_i)-path for $i \in [k]$. Note that now the digraphs may have parallel arcs and a vertex from $\{s_i, t_i\}$ may be a vertex of one or more paths P_j with $j \neq i$. Let us denote by $\mu_D(u, v)$ the number of arcs from u to v in the digraph D.

For acyclic digraphs the weak k-linkage problem is polynomial for every fixed k [7]. For digraphs for which all closed trails have length at most p (e.g. when all strong components have size at most p) we can obtain a polynomial algorithm.

Theorem 4.3. *For every natural number p the weak k-linkage problem is polynomial for digraphs containing no closed trail of length more than p.*

Proof. Let $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ be an instance of the weak k-linkage problem where D has no closed trail longer than p. Form a new digraph D' by adding $2k$ new vertices $s'_1, \ldots, s'_k, t'_1, \ldots, t'_k$ and the arcs $\{s'_i s_i | i \in [k]\} \cup \{t_i t'_i | i \in [k]\}$. Clearly $[D', s'_1, \ldots, s'_k, t'_1, \ldots, t'_k]$ is a yes-instance if and only if $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ is a yes-instance. Now let $L(D')$ be the line digraph of D', that is, $V(L(D')) = A(D')$ and $A(L(D')) = \{ab | a, b \in A(D')$ and the head of a coincides with the tail of $b\}$. It is easy to see that $[D', s'_1, \ldots, s'_k, t'_1, \ldots, t'_k]$ is a yes-instance for the weak k-linkage problem if and only if $[L(D'), s'_1 s_1, \ldots, s'_k s_k, t_1 t'_1, \ldots, t_k t'_k]$ is a yes-instance for the k-linkage problem. Since D and hence also D' has no closed trail of length more than p, the circumference of $L(D')$ is bounded by p and the claim now follows from Theorem 4.1. $\square$

Note that we cannot apply the reduction above if we replace the assumption of bounded maximum length of a closed trail by bounded circumference since the circumference of $L(D')$ may be arbitrarily large compared to that of D' (on the other hand, if D is acyclic, then so is $L(D')$ and hence (as is well known) the weak k-linkage problem for acyclic digraphs reduces to the k-linkage problem in the same class in linear time).

Problem 4.4. *What is the complexity of the weak k-linkage problem for digraphs with bounded circumference?*

When the circumference is 2 we can give a polynomial algorithm.

Theorem 4.5. *The weak k-linkage problem is polynomially solvable for every fixed k in digraphs of circumference 2.*

Proof. Like for the vertex disjoint case we will reduce an instance $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ to an equivalent instance $[D', s'_1, \ldots, s'_k, t'_1, \ldots, t'_k]$ where D' is an acyclic digraph. As above we show how to reduce the number of cycles in D successively while preserving an equivalent instance until we have an equivalent acyclic instance and then we can apply the polynomial algorithm from [7] for acyclic digraphs.
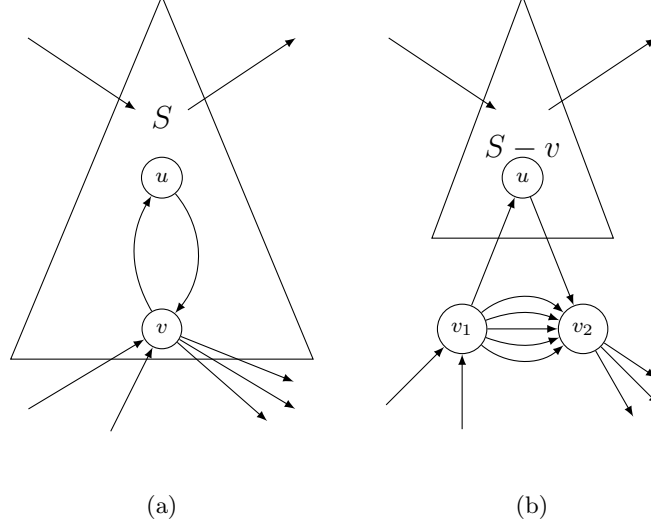

(a)   (b)

Figure 3: The operation on the graph D, to split of a leaf vertex of a non-trivial strong component.

Again we consider a non-trivial strong component S and fix a 2-cycle uvu such that u is the only neighbour of v in S (there may be several arcs in both directions between u and v). Let $d = d^-_{V-V(S)}(v) + d^+_{V-V(S)}(v)$, that is, the total number of arcs with one end in v and the other in $V - V(S)$. Let D^* be obtained from D by replacing v by two new vertices v_1, v_2, adding d arcs from v_1 to v_2 and replacing the arcs incident with v in D by new arcs incident to v_1, v_2 as follows: for every vertex $w \notin \{u, v\}$ add $\mu_D(w, v)$ arcs from w to v_1 and $\mu_D(v, w)$ arcs from v_2 to w. Finally add $\mu_D(v, u)$ arcs from v_1 to u and $\mu_D(u, v)$ arcs from u to v_2, see Figure 3

It is easy to see that no new cycle is created when going from D to D^* and the 2-cycle uvu disappears. It is also easy to verify that $[D^*, s'_1, \ldots, s'_k, t'_1, \ldots, t'_k]$ is a yes-instance if and only if $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$, where we let $s'_i = s_i$ ($t'_i = t_i$) if $s_i \neq v$ ($t_i \neq v$) and $s'_i = v_1$ if $s_i = v$ ($t'_i = v_2$ if $t_i = v$) : the d arcs between v_1 and v_2 allow us to reroute any set of paths containing v in D (including paths starting or ending here) as paths in D^* and conversely any solution in D^* can be converted back to a solution in D. In particular, if a path P'_i in D^* contains the subpath v_1uv_2 we replace this part by the vertex v.

Thus after at most $|V| - 1$ repetitions of the operation above we have converted the original instance into an equivalent acyclic instance. □

5   Concluding remarks

Contrary to the case of undirected graphs where a large class of NP-complete problems become polynomially solvable for graph of bounded tree-width, bounded DAG-width often does not lead to polynomial algorithms. For a discussion on this see [8]. The Hamiltonian cycle problem is polynomially solvable on digraphs of bounded directed tree-width [12] and hence, by Theorem 2.5 also on digraphs of bounded

DAG-width.

The minimum strong spanning subdigraph (MSSS) problem is the problem of deciding, for a given strong digraph $D = (V, A)$ a spanning strong subdigraph $D' = (V, A')$ with the minimum possible number of arcs. This problem is clearly NP-hard as for general digraphs as it contains the hamiltonian cycle problem as a special case. Khuller et al [13] showed that the MSSS problem is NP-hard already for digraphs of circumference at most 5. Hence, by Theorem 3.1, we get.

Theorem 5.1. *The MSSS problem is NP-hard for digraphs of DAG-width at most 5.*

The following Erdős-Pósa type result was proved recently by Havet and Maia.

Theorem 5.2. *[10] There exists a function $f(k)$ such that every digraph D either has k disjoint cycles of length at least 3 or a set X of size at most $f(k)$ such that $D - X$ has circumference at most 2.*

From this result and Theorem 3.1 we obtain the following by letting $t(k) = f(k) + 2$.

Corollary 5.3. *There exists a function $t(k)$ such that digraphs without k disjoint cycles of length at least 3 have DAG-width at most $t(k)$.* □

From Corollary 5.3 and Theorem 2.7 we obtain.

Corollary 5.4. *The k-linkage problem is polynomially solvable for all fixed k in the class of digraphs with at most l disjoint cycles of length at least 3.*

A similar result holds for the weak k-linkage problem but here more work is required.

Theorem 5.5. *For every choice of natural numbers k, l the weak k-linkage problem is solvable in polynomial time in the class of digraphs with no set of l disjoint cycles all of length at least 3.*

Proof. (sketch) Let $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ be an instance of the weak k-linkage problem where D is a digraph with no set of l disjoint cycles of length at least 3 (it may contain arbitrarily many disjoint 2-cycles). By Theorem 5.2 there exists a set $X \subseteq V(D)$ of size at most $f(l)$ such that $D' = D - X$ has circumference 2 and we can find such a set in polynomial time (this follows from the proof in [10] but we could also just try all subsets of size at most $f(l)$).

Now suppose $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ is a yes-instance and that $P_1, \ldots, P_k$ is a set of arc-disjoint paths forming a solution. Each P_i can visit X between 0 and $|X|$ times, the first case corresponding to the path either staying completely inside X or avoiding it altogether. We will not cover all the possibilities, but the idea should be clear from the description below. Suppose below that every P_i starts and ends in $V - X$ (that is, $\{s_1, \ldots, s_k, t_1, \ldots, t_k\} \cap X = \emptyset$) and visits X some number r_i with $0 \leq r_i \leq |X|$ times (the case when some pairs s_i, t_i intersect X is easily adapted from the case below). If $r_i = 0$ P_i is just an (s_i, t_i)-path in D' so suppose $r_i \geq 1$. Then $P_i = P_{i,1} a_{i,1} Q_{i,1} a'_{i,1} P_{i,2} a_{i,2} \ldots Q_{i,r} a'_{i,r_i} P_{i,r_i}$ where $a_{i,q} = u_{i,q} \tilde{u}_{i,q}$, $a'_{i,q} = \tilde{v}_{i,q} v_{i,q}$ are arcs respectively from $V - X$ to X and from X to $V - X$, $P_{i,1}$ is an $(s_i, u_{i,1})$-path, P_{i,r_i} is a (v_{i,r_i}, t_i)-path, $P_{i,j}$, $j \in [r_i]$, is a $(v_{i,j-1} u_{i,j})$ path all in D' and each $Q_{i,j}$, $j \in [r_i]$, is a $(\tilde{u}_{i,j}, \tilde{v}_{i,j})$-path in the subdigraph $D\langle X \rangle$ induced by X. So P_i corresponds to $r_i + 1$ arc-disjoint paths in D' and r_i arc-disjoint paths in $D\langle X \rangle$. Similarly for the remaining paths P_j. Thus the solution $P_1, \ldots, P_k$ in D corresponds to a solution to a weak r-linkage problem in D', where r represents the total number of subpaths of the P_i's that lie inside D' and a solution to a weak linkage problem in X

9

(the paths of the form $Q_{i,j}$). Clearly the converse also holds, if we have such a collection of paths in D' and $D\langle X \rangle$ and the appropriate set of distinct arcs going to and from X, then $[D, s_1, \ldots, s_k, t_1, \ldots, t_k]$ is a yes-instance. Since $D\langle X \rangle$ has bounded size, we can check any combination of paths here in constant time. Also r cannot be larger than $k|X| + k$ (since no P_i visits X more than $|X|$ times) and hence the corresponding weak r-linkage problem for D' can be solved in polynomial time via the algorithm from Theorem 4.5.

Hence by considering all possible choices of r_i arc pairs $a_{i,j}, a'_{i,j}$ for $i \in [k]$, $j \in [r_i]$ (all distinct) and solving each of the corresponding weak linkage problems we obtain a polynomial algorithm for the weak k-linkage problem (there are at most $|X|^{2k}$ different choices for the r_i's, each involving at most $2k|X|$ arcs between X and $V - X$ and hence it suffices to solve a polynomial number of weak linkage problems in D'). $\qquad\square$

References

[1] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications, 2nd Edition*. Springer-Verlag, London, 2009.

[2] J Bang-Jensen, F. Havet, and A.K. Maia. Finding a subdivision of a digraph. *Theoretical Computer Science*, 562:283–303, 2014.

[3] J. Bang-Jensen and C. Thomassen. A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM J. Discrete Math.*, 5:366–376, 1992.

[4] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdržálek. The DAG-width of directed graphs. *J. Combin. Theory, Ser B*, 102:900–923, 2012.

[5] E. Birmelé. Tree-width and circumference of graphs. *J. Graph Theory*, 43:24–25, 2003.

[6] M Chudnovsky, A Scott, and P. Seymour. Disjoint paths in tournaments. *Advances in Mathematics*, 270:582–597, 2015.

[7] S. Fortune, J.E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[8] R. Ganian, P. Hliněný, J. Kneis, A. Langer, J. Obdržálek, and P. Rossmanith. Digraph width measures in parametrized algorithmics. In *IWPEC 2009 (Eds. J. Chen and F. Fomin), Lecture Notes in Computer Science 5917*, pages 185–197, Berlin Heidelberg, 2009. Springer Verlag.

[9] R. Ganian, P. Hliněný, J. Kneis, D. Meister, J. Obdržálek, P. Rossmanith, and S. Sikdar. Are there any good digraph width measures? In *IPEC 2010 (Eds. V. Raman and S. Saurabh), Lecture Notes in Computer Science 6478*, pages 135–146, Berlin Heidelberg, 2010. Springer Verlag.

[10] F Havet and A.K Maia. On disjoint directed cycles with prescribed minimum lengths. Research report no 8286, INRIA, Sophia Antipolis, project-team COATI, 2013.

[11] P. Hunter and S. kreutzer. Digraph measures: Kelly decompositions, games and orderings. *Theoretical Computer Science*, 399:206–219, 2008.

[12] T. Johnson, N. Robertson, P.D. Seymour, and R. Thomas. Directed Tree-Width. *J. Combin. Theory Ser. B*, 82(1):138–154, 2001.

[13] S. Khuller, B. Raghavachari, and N. Young. On strongly connected digraphs with bounded cycle length. *Discrete Appl. Math.*, 69(3):281–289, 1996.

[14] J. Obdržálek. DAG-width: connectivity measure for directed graphs. In *SODA'06: Proc. 17th annual ACM-SIAM symposium on Discrete algorithm*, pages 814–821, New York, NY, USA, 2006. ACM.

[15] N. Robertson and P.D. Seymour. Graph minors. XIII: The disjoint paths problem. *J. Combin. Theory Ser. B*, 63:65–110, 1995.

6   Appendix

To illustrate how to obtain a DAG-decomposition from the strategy for the cops used in the proof of Theorem 3.1 we will now give an example to show how the DAG decomposition can be obtained by using this strategy. In figure 4a the digraph D on 32 vertices is shown. D has circumference 4 and consist of six strong components. The vertex sets of the six strong components (listed according to an acyclic ordering) are $S_1 = \{1, 2\}$, $S_2 = \{27, 28\}$, $S_3 = \{5, 6\}$, $S_4 = \{3, 4\}$, $S_5 = \{7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$ and $S_6 = \{19, 20, 21, 22, 23, 24, 25, 26, 29, 30, 31, 32\}$, and the strong component digraph $SC(D)$ (obtained by contracting each strong component into a vertex) is
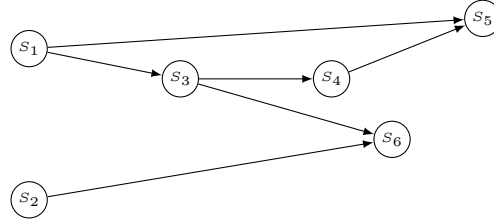
Now for each strong component we use the strategy described in the proof of Theorem 3.1. Notice that S_1, S_2, S_3 and S_4 all consist of 2 vertices and hence here, in turn, we just place a cop on each vertex and after this we have chased the robber into either S_5 or S_6. Now in $D\langle S_5 \rangle$ a maximal cycle is formed by the vertices $\{15, 16, 17, 18\}$ and we place a cop on each of these vertices. Now either the robber is in $\{7, 8, 11, 12\}$ or in $\{9, 10, 13, 14\}$. In both cases we need only keep the cop on vertex 15 to keep the robber in the same strong component. Say the robber is in $\{7, 8, 11, 12\}$ then in the next step we place a cop on 12, and note that the maximal cycle containing $\{12, 15\}$ is the 2-cycle formed by these vertices. Now the cop in vertex 15 can be lifted and placing a cop on 11 and finding a maximal cycle containing $11, 12$ gives the cycle $\{7, 8, 11, 12\}$ witch means that we have caught the robber and are done. The argument is symmetric for $\{9, 10, 13, 14\}$. The DAG-decomposition of this strong component is seen in the subtree with root $\{15, 16, 17, 18\}$ in the total DAG-decomposition in figure 4b. For the DAG-decomposition of S_6 we start by placing four cops on the maximal cycle C formed by $\{19, 20, 23, 24\}$. Placing the cops on C forces the robber to move to one of the two strong components $\{29, 30\}$ and $\{21, 22, 25, 26, 31, 32\}$ of $D - V(C)$. For the first of these the cops in vertex $\{19, 20, 23\}$ are lifted and one is placed on 29 and in next step the cop on 24 is lifted and placed on 30 and we are done. With the second strong component it is instead the cops on vertex $19, 23, 24$ that are lifted and one is placed on 21. Then the cop on 20 is

lifted and cops are placed on $22, 25, 26$. The last two steps are equivalent to what we did after covering $\{19, 20, 23, 24\}$.

Now that we have found these DAG decompositions of each of the strong components, we can combine the DAG decomposition and $SC(D)$ to obtain the DAG decomposition for the hole graph. This is seen in figure 4b
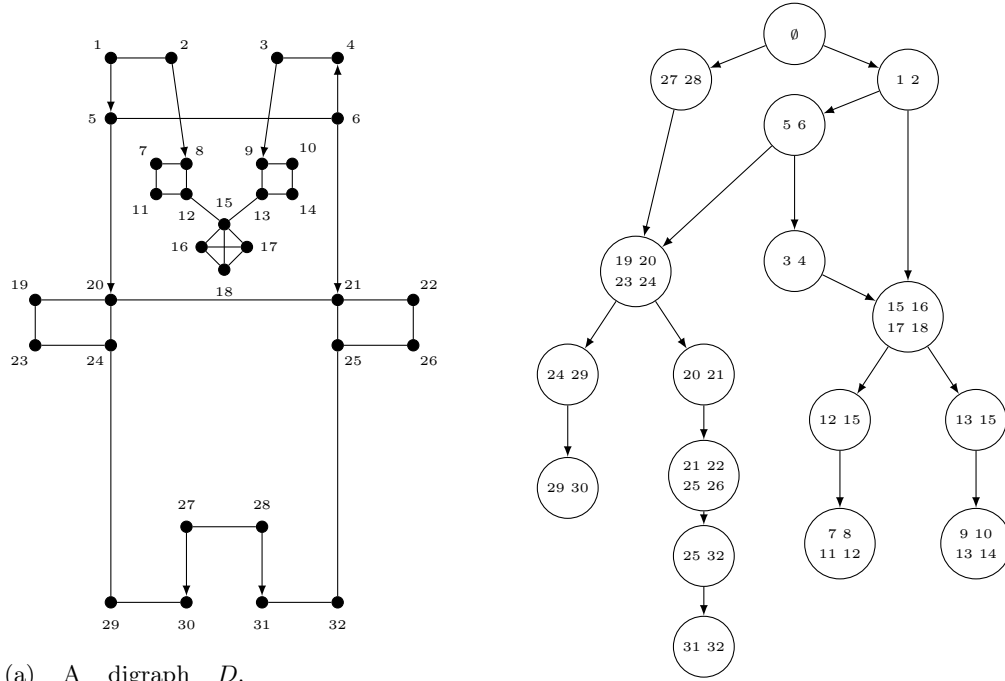
(a) A digraph D. Undirected edges correspond to 2-cycles.

(b) A DAG decomposition of D obtained via the cops and robber game.

Figure 4